香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 10: VHDL versus Verilog

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

- VHDL & Verilog: Background & Design Concept
- VHDL vs. Verilog
  - Modeling Capacity
  - Syntax & Popularity
  - Overall Structure
  - I/O Declaration
  - Concurrent Statements
  - Sequential Statements
    - Non-blocking Assignment: Combinational Logic
    - Blocking Assignment: Sequential Logic
  - Structural Design
  - Design Constructions
  - Example: Flip-flop with Synchronous Reset

What are VHDL and Verilog?

VHDL

Very High Speed Integrated Circuit
Hardware Description Language

- They are both hardware description languages for modeling hardware.
- They are each a notation to describe the behavioral and structural aspects of an electronic digital circuit.

VHDL: Background

- **V**HSIC **H**ardware **D**escription **L**anguage
  - VHSIC: Very High Speed Integrated Circuit.
- Developed by the department of defense (1981)
  - In 1986 rights where given to IEEE.
  - Became a standard and published in 1987.
  - Revised standard we know now published in 1993 (VHDL 1076-1993) regulated by VHDL international (VI).

VHDL
Very High Speed Integrated Circuit Hardware Description Language

VHDL: Design Concept

- VHDL uses top-down approach to partition design into small blocks (i.e., "**`components`**")
  - **Entity**: Describe interface signals & basic building blocks
  - **Architecture**: Describe behavior, each entity can have multiple Architectures
  - ***Configuration**: Specify different architectures for a single entity
    - The internals can change while the interface remains the same.
    - It is not required to get a basic VHDL design running.
  - ***Package**: Contain user-defined subprograms, constant definitions, and/or type definitions to be used throughout one or more design units

**: optional*

Verilog: Background

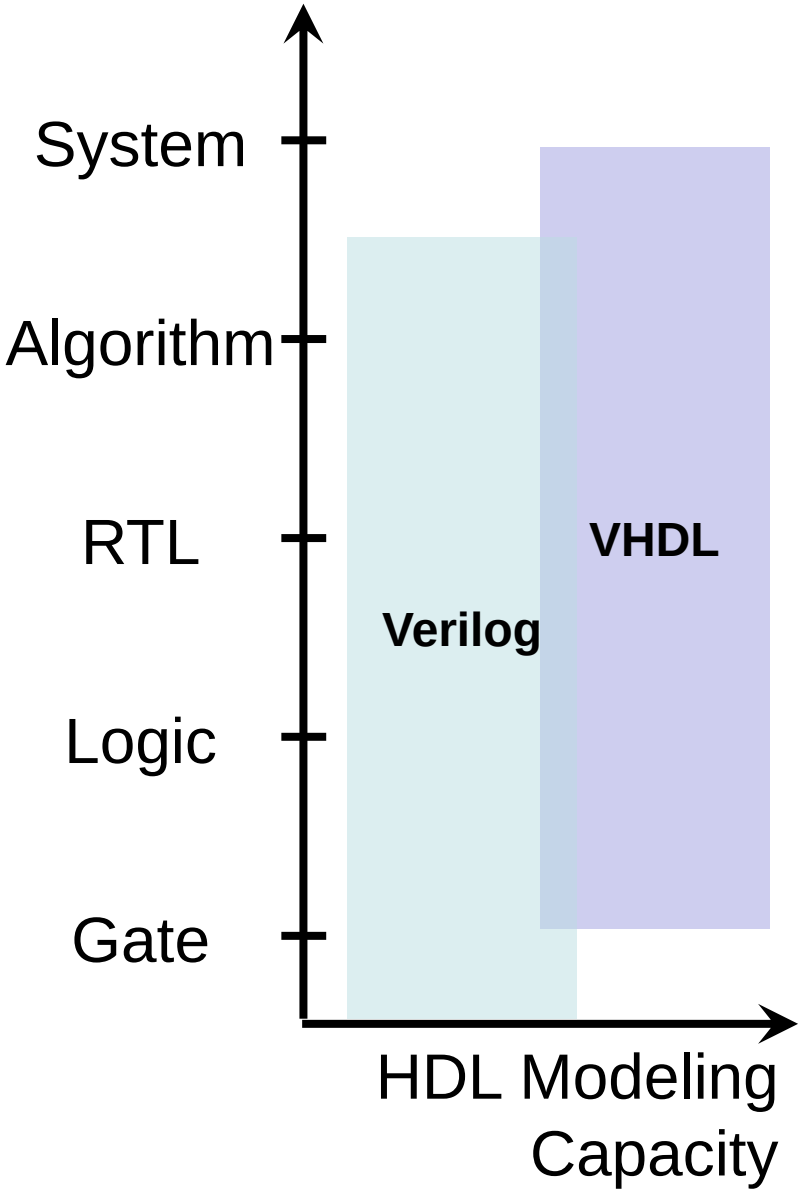
- Developed by Gateway Design Automation (1980)
  - Later acquired by Cadence Design(1989) who made it public in 1990
  - Became a standardized in 1995 by IEEE (Std 1364) regulated by Open Verilog International (OVI)

Verilog: Design Concept

- Verilog only has one building block: **`module`**
  - There is only one module per file (**`.v`**) usually.
  - Modules connect through their ports (similarly as in VHDL).
  - A top level module invokes instances of other modules.

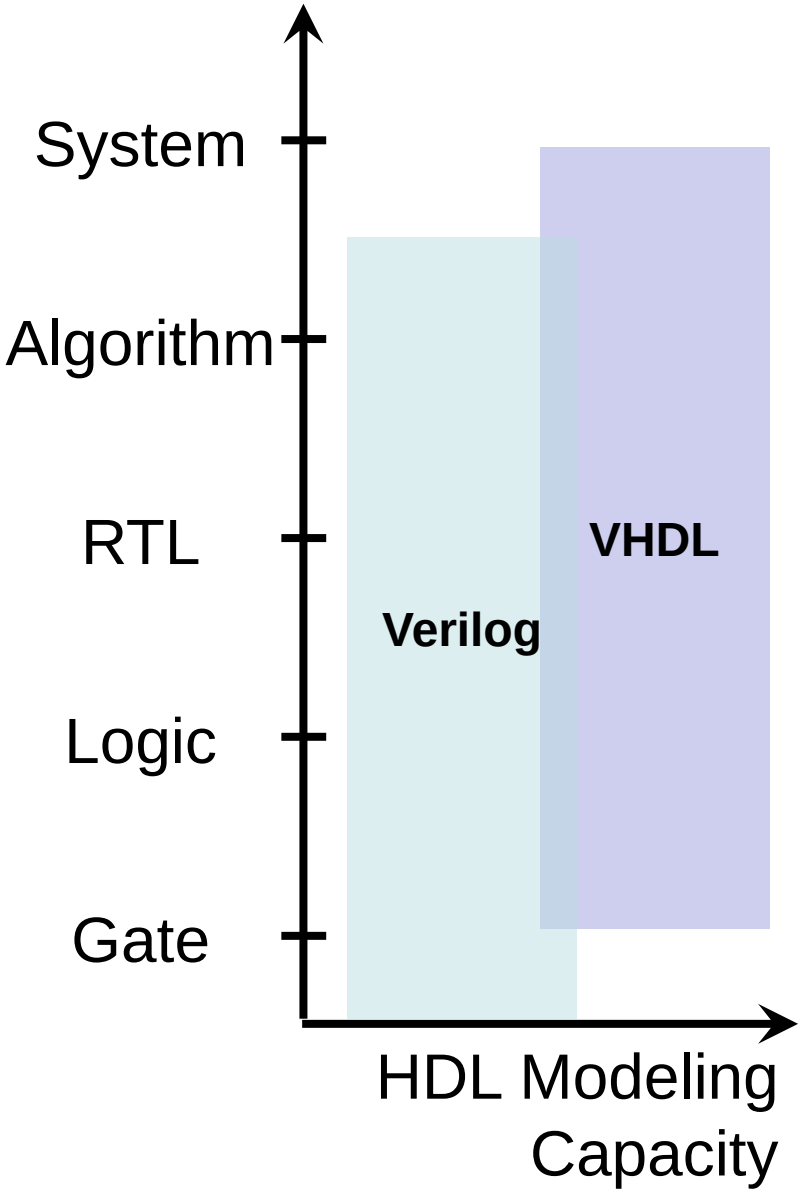
- Modules can be specified behaviorally or structurally.
  - **Behavioral**: Define behavior of digital system.
    - Similarly as the behavioral design ("**`process`**") in VHDL
  - **Structural**: Define hierarchical interconnection of modules.
    - Similarly as the structural design ("**`port map`**") in VHDL

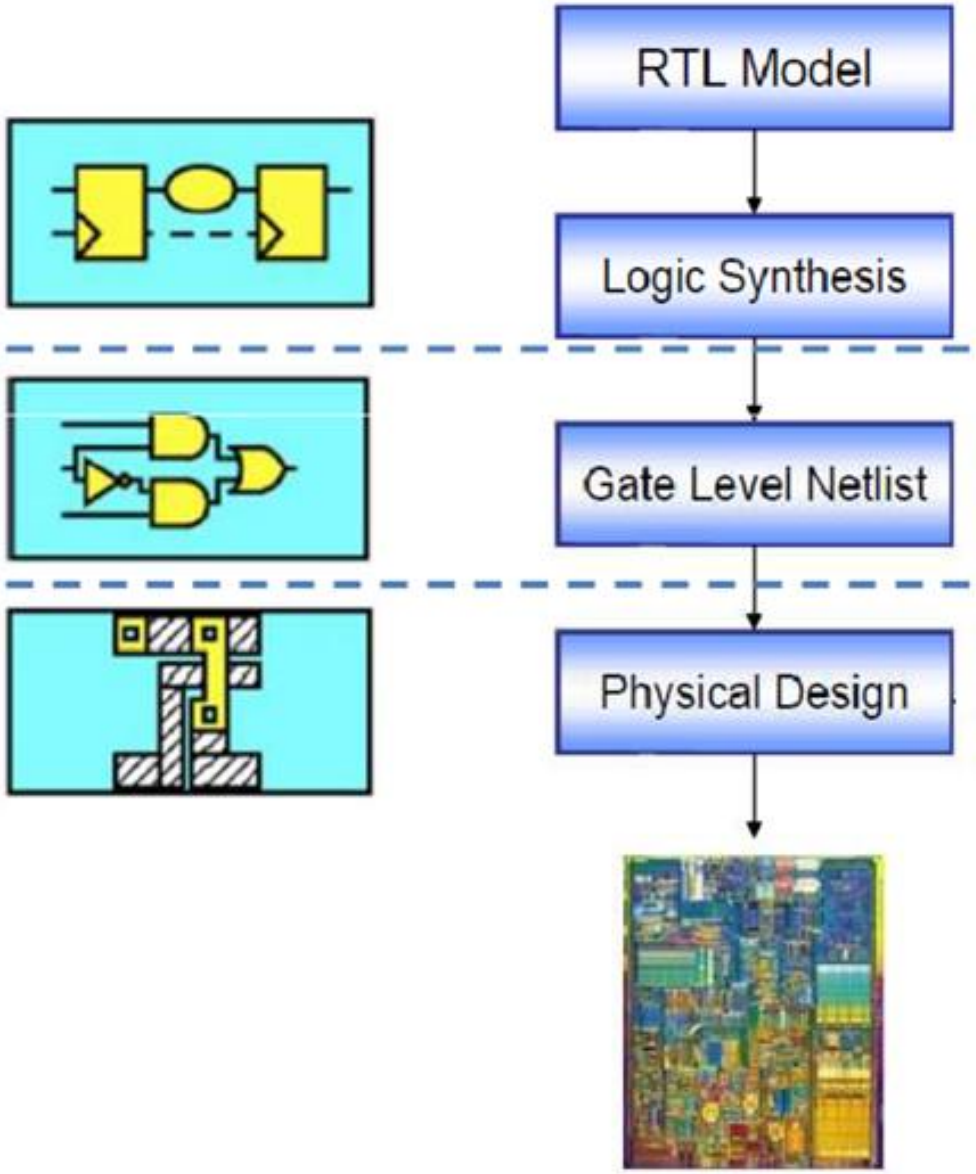
Modeling Capacity



- High-Level: VHDL is better.
  - VHDL has more features for high-level hardware modeling.
    - Such as user-defined data types, package, configuration, library management.
- Low-Level: Verilog is better.
  - Verilog is originally created for modeling logic gates.
  - Verilog has built-in primitives or low-level logic gates.
  - Verilog supports user-defined primitives (UDP).

IC Design Flow





Syntax, Popularity

VHDL
Very High Speed Integrated Circuit
Hardware Description Language

Programming Style (Syntax)	
VHDL is similar to Ada programming language.	**Verilog** is similar to C/Pascal programming language.
VHDL is NOT case-sensitive.	**Verilog** is case-sensitive.
VHDL is more "verbose" than **Verilog**.	

Popularity	
VHDL is more popular with European companies.	**Verilog** is more popular with US companies.

Overall Structure

VHDL

```
-- Library Declaration
library IEEE;
…
-- Entity Declaration
entity mux is
…
end mux
-- Architecture Body
architecture arch of mux is
begin
…
end arch;
```

Verilog

```
// One Module
module mux (a, b, s, y);
…
endmodule
```

I/O Declaration

VHDL

```
-- Library Declaration
…
-- Entity Declaration
entity mux is
port(a,b,s: in std_logic;
     y: out std_logic);
end mux
-- Architecture Body
architecture arch of mux is
begin
…
end arch;
```

Verilog

```
// One Module
module mux (a, b, s, y);
input a,b,s;
output y;
…
endmodule
```

Concurrent Statement: assign

VHDL: *inside architecture body, outside the process*

```
signal a, b: std_logic_vector(7 downto 0);
signal c, d, e: std_logic;
a(3 downto 0) <= b(7 downto 4);
b <= "00001010";
c <= d and e;
```

Verilog: *outside the always block*

```
wire [7:0] a, b;
wire c, d, e;
assign a[3:0] = c[7:4];
assign a[7:4] = 'b0000; // binary
assign a = b & c; // logical AND
```

assign:

- Assignments outside an **always** block are concurrent.
- LHS must be **wire**.
- The LHS will be updated whenever RHS changes.

Class Exercise 10.1

Student ID: _____ Date:
Name: _____ _____

- Translate the following VHDL program to Veirlog:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity abc is
  port (a,b,c: in std_logic;
        y: out std_logic);
end abc;
architecture abc_arch of
abc is
signal x : std_logic;
begin
  x <= a nor b;
  y <= x and c;
end abc_arch;
```

Sequential Statement: = or <=

VHDL

```
architecture arch of ex is
begin
process (…)
begin
   -- LHS could be signals or
      variables
   -- signal assignment (<=)
   --              or
   -- variable assignment (:=)
end;
end arch;
```

Verilog

```
module ex (…);
reg a, b, c;
always @(…)
begin
   // LHS must be reg (not wire)
   // blocking assignment (=)
      for combinational logic
   //              or
   // non-blocking assignment
      (<=) for sequential logic
end
endmodule
```

Blocking Assignment: Combinational Logic

- Blocking Assignment (=): *Executed before the execution of the statements in a sequential block (e.g., always block).*
  - A blocking assignment (=) will evaluate the RHS and perform the LHS assignment immediately.
    - The second assignment is performed, once the first one is complete.
    - The assignments are executed in a sequential way.
- **Usage**: Use blocking assignment (=) for always blocks that are purely combinational logic.

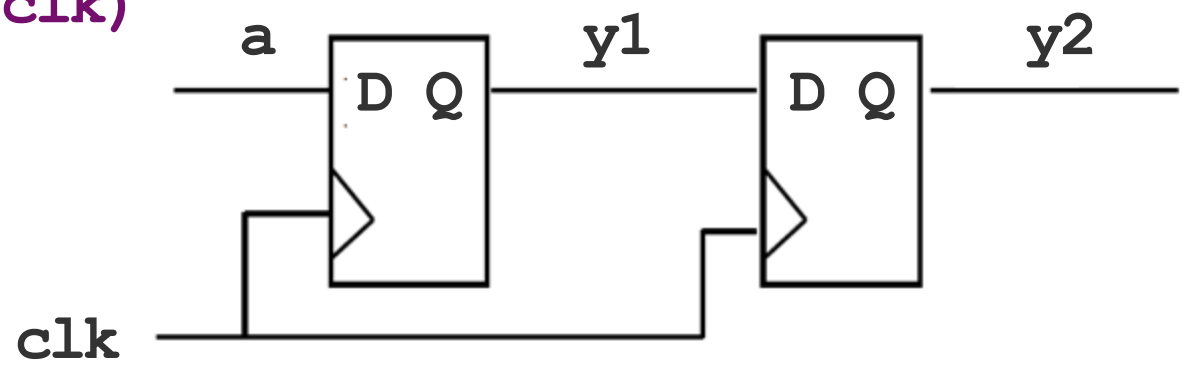
```
reg t1, t2, t3;
always @(a or b) begin
   t1 = a & b; // logical AND
   t2 = c | d; // logical OR
   t3 = t1 | t2;
end
```

Non-blocking Assignment: Sequential Logic

- Non-blocking Assignment (<=): *Schedule assignments without blocking the procedural flow.*
  - A non-blocking assignment (<=) samples RHS at the beginning of a clock edge; with the actual LHS assignment taking place at the end of that clock edge.

- **Usage**: Use non-blocking assignments (<=) in always blocks to synthesize/simulate sequential logic.

```
reg y1, y2;
always @(posedge clk)
begin
   y1 <= a;
   y2 <= y1;
end
```

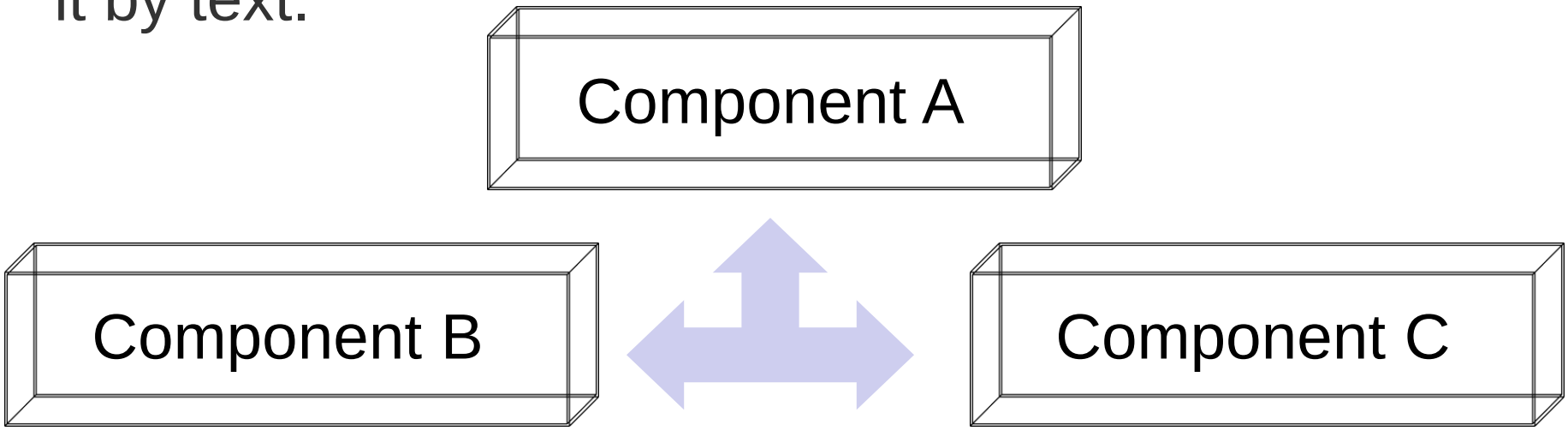


"wire" vs. "reg" in Verilog

- **Wire**: Has ***no*** memory
  - Physical wire in the circuit
  - A wire does not store its value, it must be driven by
    - Connecting the wire to the output of a gate or module, or
    - Assigning a value to the wire in a concurrent assignment
  - **Usage:** Cannot use "**`wire`**" in left-hand-side of assignments in always blocks.
- **Register**: Has memory
  - Not "register" of CPU
  - Could be a flip-flop (DFF) or a physical wire
  - Holding value until a new value is assigned
  - **Usage:** Cannot use "**`reg`**" in left-hand side of assignments outside always blocks (i.e., concurrent assignment)

Structural Design in VHDL (1/2)

- Structural Design in VHDL: Like a circuit but describe it by text.



Connected by **`port map`** in architecture body

- Design Steps:
  - Step 1: Create **`entities`**
  - Step 2: Create **`components`** from **`entities`**
  - Step 3: Use "**`port map`**" to relate the components

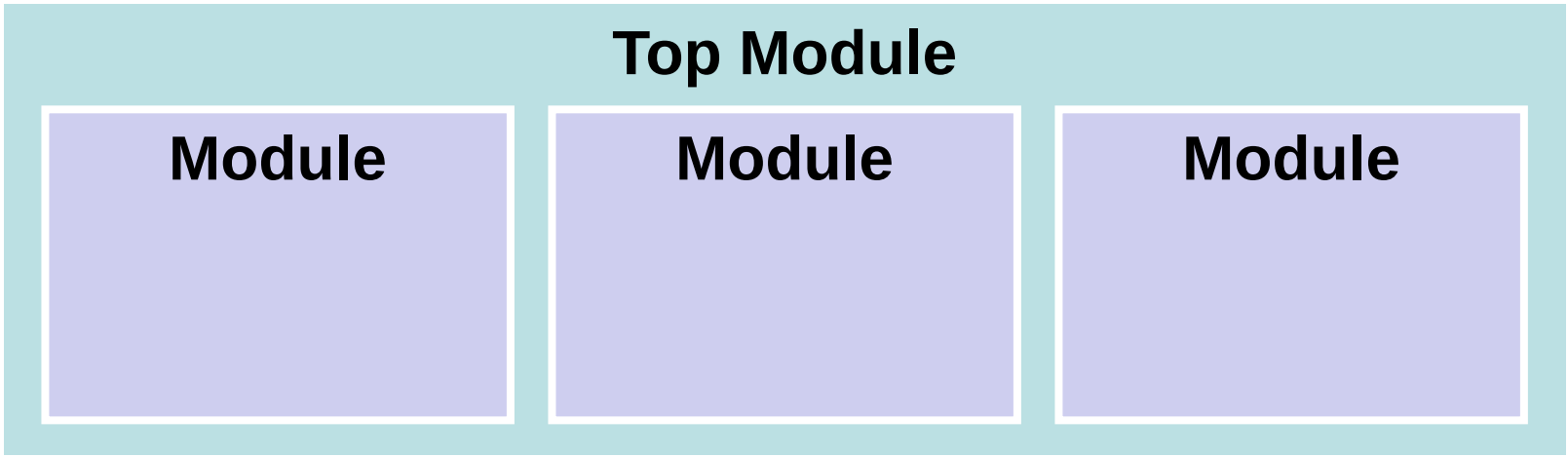
Structural Design in VHDL (2/2)

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 entity and2 is                          Step 1
 4 port (a,b: in STD_LOGIC;
 5         c: out STD_LOGIC );
 6 end and2;
 7 architecture and2_arch of and2 is
 8 begin
 9   c <= a and b;
10 end and2_arch;
11 ------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 entity or2 is                           Step 1
15 port (a,b: in STD_LOGIC;
16         c: out STD_LOGIC );
17 end or2;
18 architecture or2_arch of or2 is
19 begin
20   c <= a or b;
21 end or2_arch;
```

```
 1 library IEEE;
 2 use IEEE.STD_LOGIC_1164.ALL;
 3 ----------------------------------------
 4 entity test is
 5 port ( in1: in STD_LOGIC; in2: in STD_LOGIC;
 6          in3: in STD_LOGIC;
 7        out1: out STD_LOGIC );
 8 end test;
 9 architecture test_arch of test is
10 component and2 --create component          Step 2
11   port (a,b: in std_logic; c: out std_logic);
12 end component ;
13 component or2 --create component
14   port (a,b: in std_logic; c: out std_logic);
15 end component ;
16 signal inter_sig: std_logic;
17 begin                                      Step 3
18   label1: and2 port map (in1, in2, inter_sig);
19   label2: or2 port map (inter_sig, in3, out1);
20 end test_arch;
```

Structural Design in Verilog (1/2)

- Structural Design in Verilog: One top module, several (sub) modules.



Connected by relating I/O and internal wires

- Design Steps:
  - Step 1: Create (sub) **`module(s)`** (usually in separate **`.v`** files)
  - Step 2: Define a **`top-module`** to interconnect **`module(s)`**

Structural Design in Verilog (2/2)

and2.v

Step 1

```
module and2(
  input a,
  input b,
  output c
  );
  assign c = a && b;
endmodule
```

or2.v

Step 1

```
module or2(
  input a,
  input b,
  output c
  );
  assign c = a || b;
endmodule
```

top_module.v

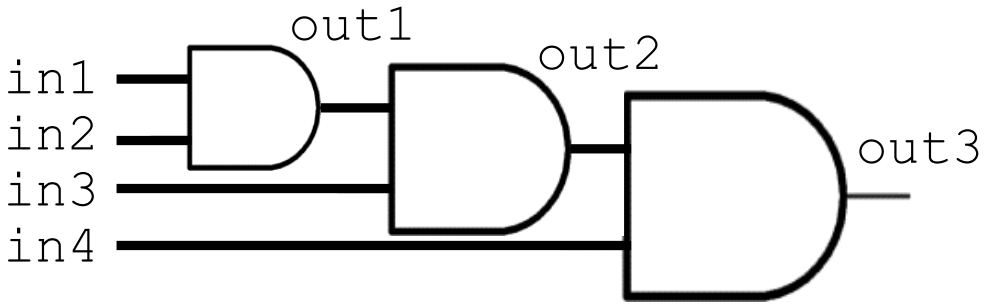
Step 2

```
module top_module(
  input in1, input in2, input in3,
  output out1 );
  wire inter_sig;
  and2 and2_ins(
    .a(in1),
    .b(in2),
    .c(inter_sig)
  );
  or2 or2_ins(
    .a(inter_sig),
    .b(in3),
    .c(out1)
  );
endmodule
```

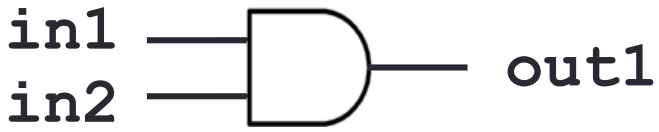
Class Exercise 10.2

Student ID: _____ Date:
Name: _____ _____

- Implement the following circuit in Veirlog:



Design Constructions (1/4)



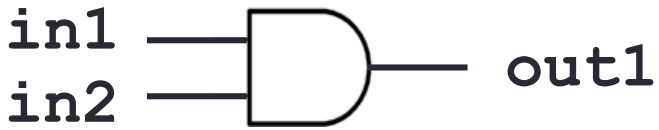
VHDL: `when-else` *(concurrent, outside process)*

```
architecture arch of ex is
begin
   out1 <= '1' when in1 = '1' and in2 = '1' else '0';
end arch ex_arch;
```

Verilog: `assign ? :` *(concurrent, outside always@ block)*

```
module ex (…);
   assign out1 = (in1=='b1 && in2=='b1) ? 'b1 : 'b0;
   // 'b: binary; 'o: octal; 'd: decimal; 'h: hexadecimal
endmodule
```

Design Constructions (2/4)



VHDL: `if-then-else`

(sequential, inside process)

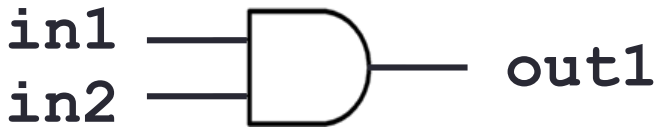
```
process(in1, in2)
begin
  if in1='1' and in2='1'
  then
    out1 <= '1';
  else
    out1 <= '0';
  end if;
end process;
```

Verilog: `if-else`

(sequential, inside always@)

```
always @(in1, in2)
begin
  if (in1=='b1 && in2=='b1)
  begin
      out1 = 'b1;
  end
  else
  begin
    out1 = 'b0;
  end
end
```

Design Constructions (3/4)



VHDL: case-when

(sequential, inside process)

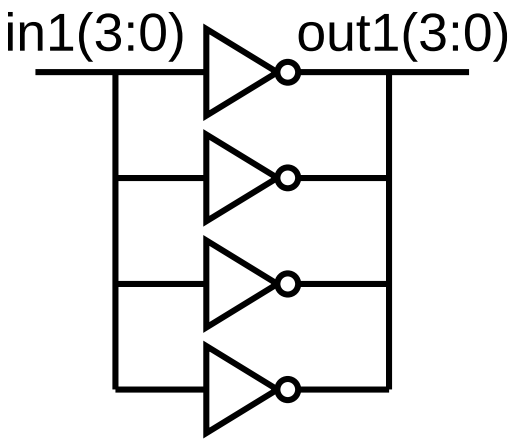
```
process(b)
begin
  case b is
    when "00"|"11" =>
      out1 <= '0';
      out2 <= '1';
    when others    =>
      out1 <= '1';
      out2 <= '0';
  end case;
end process;
```

Verilog: case

(sequential, inside always@)

```
always @(b)
begin
  case (b)
    'b00 || 'b11:
      out1 = 'b0;
      out2 = 'b1;
    default:
      out1 = 'b1;
      out2 = 'b0;
  endcase
end
```

Design Constructions (4/4)



VHDL: `for-in-to-loop`
(sequential, inside process)

```
process(in1)
begin
  for i in 0 to 3 loop
    out1(i) <= not in1(i);
  end loop;
end process;
```

Verilog: `for-loop`
(sequential, inside always@)

```
always @(in1)
begin
  for(idx=0; idx<4; idx+=1)
  begin
    out1[idx] = ~in1[idx];
  end
end
```

Example: Flip-flop with Sync. Reset

VHDL

```
entity dff is
port(D,CLK,RESET:
     in std_logic;
     Q: out std_logic);
end dff;
architecture dff_arch of
dff is begin
  process(CLK) begin
    if rising_edge(CLK) then
      if (RESET = '1') then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end dff_arch;
```

Verilog

```
module dff(
    input D,
    input CLK,
    input RESET,
    output reg Q); // why?
always @(posedge CLK)
begin

  if (RESET) begin
    Q <= 1'b0; // non-blocking
  end
  else begin
    Q <= D; // non-blocking
  end

end
endmodule
```

```
always @(...)
begin
  // LHS must be reg (not wire)
```

Example: Flip-flop with Async. Reset

VHDL

```
entity dff is
port(D,CLK,RESET:
     in std_logic;
     Q: out std_logic);
end dff;
architecture dff_arch of
dff is begin
  process(CLK) begin
    if rising_edge(CLK) then
      if (RESET = '1') then
        Q <= '0';
      else
        Q <= D;
      end if;
    end if;
  end process;
end dff_arch;
```

Verilog

```
module dff(
    input D,
    input CLK,
    input RESET,
    output reg Q); // why?
always @(posedge CLK or
         posedge RESET)
begin
  if (RESET) begin
    Q <= 1'b0; // non-blocking
  end
  else begin
    Q <= D; // non-blocking
  end

end
endmodule
```

```
always @(...)
begin
  // LHS must be reg (not wire)
```

Summary

- VHDL & Verilog: Background & Design Concept
- VHDL vs. Verilog
  - Modeling Capacity
  - Syntax & Popularity
  - Overall Structure
  - I/O Declaration
  - Concurrent Statements
  - Sequential Statements
    - Non-blocking Assignment: Combinational Logic
    - Blocking Assignment: Sequential Logic
  - Structural Design
  - Design Constructions
  - Example: Flip-flop with Synchronous Reset